STRUKTUR DATA NON LINEAR

"BINARY SEARCH TREE"

MODUL PERTEMUAN KE - 3 : CREATE, INSERT, SEARCH

DISUSUN OLEH :

Nama : Andreas Nathanael Priambodo

NIM : 215314043

TEKNIK INFORMATIKA FAKULTAS SAINS DAN TEKNOLOGI UNIVERSITAS SANATA DHARMA

YOGYAKARTA 2023

A. Algoritma

- Insert

  Masukan data ke dalam object tr
  Jika root sama dengan null maka root sama dengan tr
  Jika tidak object trB sama dengan root
  Ketika benar
  Jika data kurang dari sama dengan object trB maka
  Jika node kiri kosong maka masukan tr ke kiri
  Jika tidak trB sama dengan data kiri
  Jika tidak
  Jika node kanan kosong maka masukan tr ke kanan
  Jika tidak trB sama dengan data kanan

- Pre Order

  jika node tidak sama dengan null maka
  cetak node
  ambil node kiri
  ambil node kanan

- In Order

  jika node tidak sama dengan null maka
  ambil node kiri
  cetak node
  ambil node kanan

- Post Order

  ambil node kiri
  ambil node kanan
  cetak node

B. Program

MainTree

```
package binary_search_tree_copy;


import java.util.Scanner;
```

```
public class MainTree {


    public static void main(String[] args) {
        Scanner dtSc = new Scanner(System.in);
        Tree Ob = new Tree();
        Ob.Insert(42);
        Ob.Insert(21);
        Ob.Insert(38);
        Ob.Insert(27);
        Ob.Insert(71);
        Ob.Insert(82);
        Ob.Insert(55);
        Ob.Insert(63);
        Ob.Insert(6);
        Ob.Insert(2);
        Ob.Insert(40);
        Ob.Insert(12);
        System.out.println();
        System.out.println("PreOrder : ");
        Ob.preOrderTransversal();
        System.out.println();
        System.out.println("InOrder : ");
        Ob.inOrderTransversal();
        System.out.println();
        System.out.println("PostOrder : ");
        Ob.postOrderTransversal();
        System.out.println();
        System.out.print("Search Data : ");
        int search = dtSc.nextInt();
        TreeNode dataSearch = Ob.Search(search);
        if (dataSearch == null) {
            System.out.println("Data not Found");
```

```
        } else {
            System.out.println("Data " + dataSearch.getData() + " Found");
        }
    }
}
```

Tree

```
package binary_search_tree_copy;

public class Tree {

    private TreeNode root;

    public Tree() {
        root = null;
    }

    public Tree(TreeNode root) {
        this.root = root;
    }

    public TreeNode getRoot() {
        return root;
    }

    public void setRoot(TreeNode root) {
        this.root = root;
    }

    public void preOrderTransversal() {
        preOrderHelper(root);
    }
```

```
public void inOrderTransversal() {
    inOrderHelper(root);
}

public void postOrderTransversal() {
    postOrderHelper(root);
}

public void preOrderHelper(TreeNode node) {
    if (node != null) {
        System.out.print(node.getData()+" ");
        preOrderHelper(node.getLeftNode());
        preOrderHelper(node.getRightNode());
    }

}

public void inOrderHelper(TreeNode node) {
    if (node != null) {
        inOrderHelper(node.getLeftNode());
        System.out.print(node.getData()+" ");
        inOrderHelper(node.getRightNode());
    }

}

public void postOrderHelper(TreeNode node) {
    if (node != null) {
        postOrderHelper(node.getLeftNode());
        postOrderHelper(node.getRightNode());
        System.out.print(node.getData()+" ");
```

```
    }


}

public void Insert(int in) {
    TreeNode tr = new TreeNode(in);
    if (root == null) {
        root = tr;
        System.out.println("Root " + root.getData());
    } else {
        TreeNode trB = root;
        while (true) {
            if (in <= trB.getData()) {
                if (trB.getLeftNode() == null) {
                    trB.setLeftNode(tr);
                    System.out.println(tr.getData() + " LEFT " + trB.getData());
                    break;
                } else {
                    trB = trB.getLeftNode();
                }
            } else {
                if (trB.getRightNode() == null) {
                    trB.setRightNode(tr);
                    System.out.println(tr.getData() + " RIGHT " + trB.getData());
                    break;
                } else {
                    trB = trB.getRightNode();
                }
            }
        }
    }
}
```

```
    public TreeNode Search(int cari) {
        TreeNode data = root;
        while (data != null) {
            if (cari == data.getData()) {
                return data;
            } else if (cari < data.getData()) {
                data = data.getLeftNode();
            } else {
                data = data.getRightNode();
            }
        }
        return null;
    }
}
```

TreeNode

```
package binary_search_tree_copy;

public class TreeNode {

    private int data;
    private TreeNode leftNode;
    private TreeNode rightNode;

    public TreeNode(int data) {
        this.data = data;
        leftNode = rightNode = null;
    }

    public int getData() {
        return data;
```

```
    }

    public void setData(int data) {
        this.data = data;
    }

    public TreeNode getLeftNode() {
        return leftNode;
    }

    public void setLeftNode(TreeNode leftNode) {
        this.leftNode = leftNode;
    }

    public TreeNode getRightNode() {
        return rightNode;
    }

    public void setRightNode(TreeNode rightNode) {
        this.rightNode = rightNode;
    }
}
```

C. Output

```
run:
Root 42
21 LEFT 42
38 RIGHT 21
27 LEFT 38
71 RIGHT 42
82 RIGHT 71
55 LEFT 71
63 RIGHT 55
6 LEFT 21
2 LEFT 6
40 RIGHT 38
12 RIGHT 6

PreOrder :
42 21 6 2 12 38 27 40 71 55 63 82
InOrder :
2 6 12 21 27 38 40 42 55 63 71 82
PostOrder :
2 12 6 27 40 38 21 63 55 82 71 42
Search Data :
```

```
run:
Root 42
21 LEFT 42
38 RIGHT 21
27 LEFT 38
71 RIGHT 42
82 RIGHT 71
55 LEFT 71
63 RIGHT 55
6 LEFT 21
2 LEFT 6
40 RIGHT 38
12 RIGHT 6

PreOrder :
42 21 6 2 12 38 27 40 71 55 63 82
InOrder :
2 6 12 21 27 38 40 42 55 63 71 82
PostOrder :
2 12 6 27 40 38 21 63 55 82 71 42
Search Data : 40
Data 40 Found
BUILD SUCCESSFUL (total time: 24 seconds)
```

```
run:
Root 42
21 LEFT 42
38 RIGHT 21
27 LEFT 38
71 RIGHT 42
82 RIGHT 71
55 LEFT 71
63 RIGHT 55
6 LEFT 21
2 LEFT 6
40 RIGHT 38
12 RIGHT 6

PreOrder :
42 21 6 2 12 38 27 40 71 55 63 82
InOrder :
2 6 12 21 27 38 40 42 55 63 71 82
PostOrder :
2 12 6 27 40 38 21 63 55 82 71 42
Search Data : 1
Data not Found
BUILD SUCCESSFUL (total time: 2 seconds)
```

D. Analisa

- MainTree

Pada class MainTree ini terdapat Scanner yang digunakan untuk menginput kan data ke dalam console yang sebelumya sudah diimportkan library dari scanner didalam java.util lalu setelahnya ada penginisialisasian object baru untuk class Tree yang Bernama Ob lalu dibawah nya ada pemanggilan method yang ada didalam Ob untuk insert yang digunakan utnuk memasukan data kedalam Tree nya melalui Node lalu dibawahnya pada baris ke 23 terdapat print ke terminal “Search Data : ” lalu inputan yang dimasukan kedalam variable search lalu dibawahnya terdapat variable dari dataSearch yang diisikan dengan Ob.search yang diisikan dengan search variable yang telah diinputkan sebelumnya yang digunakan untuk mencari data yang ada didalam object dari Ob lalu dataSearch tadi diubah menjadi kebentuk if else untuk menentukan apakah data yang dicari ada atau tidak jika ada maka akan mengeluarkan ke terminal Data not Found sedangkan jika ada akan menampilkan Data lalu menggunakan variable dari dataSearch.getData untuk memanggil datanya Found. Dan dibagian bawah dari inputan angka untuk tree nya terdapat pemanggilan method untuk preOrderTransversal() lalu ada

inOrderTransversal() ada juga postOrderTransversal() dimana masing masing ini digunakan untuk mengunjungi node node pada binary tree.

- Tree

  Didalam class tree ini ada attribute yang Bernama root yang berbentuk private yang dimana root ini bertipekan data class TreeNode lalu dibawahnya terdapat constructor yang digunakan untuk pendeklarasian root ini null dan dibawahnya lagi ada kontruktor lagi yang akan terisi apabila akan menambah TreeNode yang sudah ada di main kelas sehingga nanti dapat menggunakan root ini selanjutnya ada setter dan getter dari root dan juga ada method insert dan juga method search untuk inster sendiri berisikan parameter integer Bernama in dan untuk search Sendiri juga menggunakan integer juga tetapi Bernama cari untuk Search akan melakukan algoritma dari Search dan untuk Inser juga akan melakukan algoritma dari Search. Ada tahmbahan untuk sekarang terdapat pre order, in order, dan post order transversal method yang nanti dia memanggil method lain untuk method helper dari masing masing order. Yang pertama untuk preOrderHelper yang berisikan parameter node yang bertipekan class TreeNode didalam bodynya mengecek apakah node nya kosong dan jika kosong akan langsung dikembalikan ke method atasnya yaitu untuk transversal lalu akan Kembali lagi ke main classnya yaitu MainTree selanjutnya untuk bawahnya lagi ada print untuk mencetak data node nya lalu dibawah ada pemanggilan method lagi dan dia memanggil dirinya sendiri atau istilahnya dengan cara rekursif dia akan berulang kali mencari untuk melihat apakah ada leftNode atau tidak dan djika tidak akan Kembali ke rekursif awalan tadi begitu seterusnya. Sama seperti sebelumnya untuk inOrder dengan postOrder yang berbeda hanya dalam urutan print dari node nya karena print disini bisa dianggap sebagai root nya yang akan langsung dicetak.

- TreeNode

  Didalam TreeNode ini berisikan 3 attribut untuk data yang bertipekan integer lalu ada leftNode dan rightNode yang bertipekan TreeNode lalu ada 1 buah contructor yang berparameter data unguk mengisikan attribute data dan juga ada penginisialisasian untuk leftNode dan juga rightNode menjadi null selanjutnya ada setter dan getter untuk data dan juga leftNode dan rightNode